Cycle de vie du logiciel

Bruno Bouzy

Juin 2001

1. INTRODUCTION

1.1. But du document

Ce document rappelle les principes de base d'une méthode classique de développement au §2, décrit le contenu de chacune des phases du développement au §3, donne quelques informations sur la phase de maintenance au §4, et enfin propose des plans de documents associés à la méthode au §5.

1.2. Documents de référence

- SADT cours de l'IGL

- Modélisation objet avec UML, Pierre-Alain Muller, Eyrolles, 1997

2. GENERALITES SUR UNE METHODE CLASSIQUE DE DEVELOPPEMENT DE LOGICIELS

2.1. Domaine d'application d'une méthode classique de développement

Dans le cheminement de la pensée qui va de la naissance d'un projet informatique jusqu'à son terme on peut distinguer à priori deux grandes étapes qu'il est nécessaire de dissocier pour la compréhension de ce document:

La première, très en amont, ressemble à une recherche ou à une étude de faisabilité du projet. L'utilisateur a seulement une idée floue de son besoin et surtout ne sait pas comment y répondre. Il ne sait pas quelle solution est faisable. Il veut mettre en oeuvre les idées qui naissent dans sa tête pour pouvoir les valider ou les invalider. Dans les entreprises, ce sont (en schématisant) les départements d'études ou de recherche qui effectuent ce genre de travail. Les systèmes experts y sont en général utilisés car adaptés au besoin.

La seconde, plus en aval, est un développement plus concret qui ne pose pas de problème de faisabilité. Il est possible de définir clairement le besoin auquel le logiciel répond. On sait d'avance qu'il existe une solution faisable pour réaliser le logiciel. Le savoir-faire dans ce domaine peut être formalisé. Dans les entreprises, ce sont (en schématisant toujours) les départements de développement de logiciels opérationnels qui effectuent ce genre de travail.

Une méthode classique de développement s'applique à la deuxième étape (sans qu'il soit interdit d'en utiliser quelques principes dans la première étape). Ce document présente les principes de base d'une méthode classique de développement.

2.2. Critères de qualité du logiciel

Le but du développement de logiciel est de développer des logiciels de qualité. Le terme « qualité » est assez vague, grosso modo il signifie que l'on cherche a développer un logiciel qui correspond aux besoins d'un utilisateur de ce logiciel.

Pour être plus précis, il existe des « critères » de qualité qui permettent de définir différents types de qualité. Un développement peut être fait pour satisfaire tout ou partie de l'ensemble de ces critères.

Exactitude
Aptitude d'un logiciel à fournir des résultats voulus dans les conditions normales d'utilisation.
Le plus important des critères de qualité. C'est la base de l'informatique : on souhaite développer des logiciels qui répondent aux besoin de l'utilisateur.

Robustesse
Aptitude à bien réagir lorsque l'on s'écarte des conditions normales d'utilisation.
Exemple : IP (Internet Protocol). Le succès d'Internet est du à la robustesse du protocole de communication utilisé. Un datagramme IP arrive a destination même si un réseau local est inaccessible.

Extensibilité
Facilité avec laquelle un programme pourra être adapté pour faire face à une évolution des besoins de l'utilisateur.

Réutilisabilité
Possibilité d'utiliser certaines parties du logiciel pour développer un autre logiciel répondants à d'autres besoins. Cette notion est souvent relié à l'orienté objet où une classe générale sera facilement réutilisable.

Portabilité
Facilité avec laquelle on peut exploiter un logiciel dans différentes implémentations. Exemple Windows 95 ou Linux.

Efficience
Temps d'exécution, taille mémoire…

Ces critères de qualité sont des objectifs qu'un utilisateur va spécifier éventuellement dans l'expression de ses besoins.

Un méthode de développement permet de faciliter la satisfaction des critères de qualité.

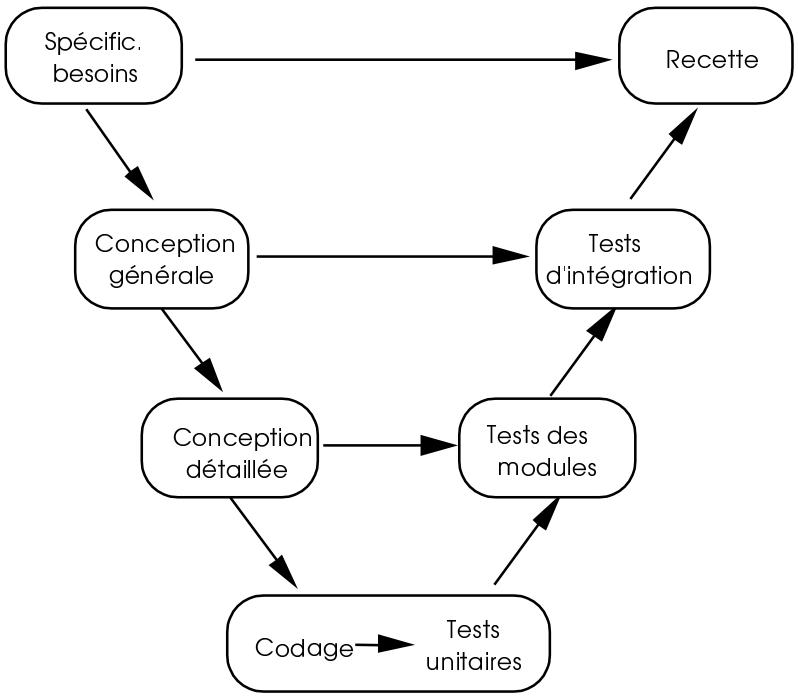
2.3. Cycle de vie

Le développement d'un logiciel se fait suivant un cycle appelé le cycle de vie du logiciel. Le cycle de vie est décomposé en phases de développement :

- Spécifications des besoins,
- Conception générale,
- Conception détaillée,
- Codage et tests unitaires,
- Intégration des modules,
- Intégration du logiciel,
- Recette.

Ces phases sont échelonnées dans le temps. Une phase se termine par la remise d'un (ou plusieurs) document(s) validé(s) conjointement par l'utilisateur et le développeur. Une phase de développement se termine lorsque la revue de cette phase est faite. Une phase ne peut commencer que lorsque la précédente est terminée. A la fin de chaque phase, l'utilisateur et le développeur sont d'accord. La décomposition en phases de développement permet donc le suivi du projet par l'utilisateur.

Schéma représentant le processus de développement d'un logiciel classique :



Les premières phases permettent de décomposer l'ensemble du projet pour simplifier la phase de codage (Top-Down). Les phases suivantes recomposent l'ensemble du logiciel en le testant du détail vers l'ensemble (Bottom-Up).

2.3. La documentation

Au cours d'un projet, la qualité première que l'on doit attendre des personnes qui interviennent de près ou de loin dans ce projet, est la volonté de communiquer avec les autres. Si tous les intervenants sont animés par cette volonté commune, la réussite du projet est très largement favorisée. Sinon, l'echec est certain.

L'information échangée entre les participants contient deux parties. Une partie est un échange d'informations informel qui n'a pas besoin d'être structuré et qui sert à créer une ambiance de travail, une motivation , etc... Cette information est en générale transmise oralement. Une autre partie est un échange d'information plus technique qui doit être structuré. La documentation permet de répondre à ce besoin.

L'objectif de la documentation est de permettre la transmission de l'information, rendre visible le produit logiciel tout au long du cycle de vie. En particulier à la fin du projet, la documentation qui intéresse les utilisateurs du logiciel est prête.

C'est le seul produit du développement lors des premières phases de développement.

La documentation sert de support de travail, assure la conservation et la transmission de la connaissance, facilite la conduite, la gestion et le contrôle du projet.

La documentation est produite au fur et à mesure du développement du projet. Les documents validés à la fin de chaque phase constituent des jalons et servent de base de travail pour les activités de la phase suivante. En particulier, cela permet à des personnes extérieures au projet d'être mises au courant rapidement lorsqu'elles arrivent en cours de projet.

La documentation est organisée suivant un plan de documentation qui peut être adapté en fonction du projet lui-même.

3. LES PHASES DU DEVELOPPEMENT

3.1. Phase de spécifications des besoins

3.1.1. Objectif

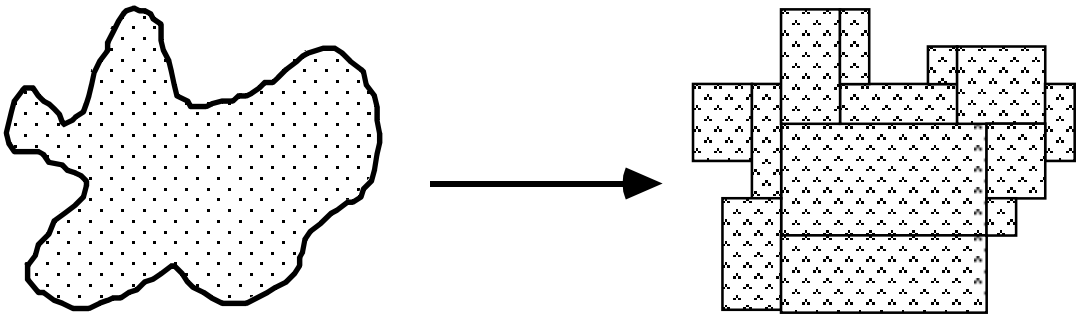
Il est indispensable de déterminer les besoins du logiciel pendant la première phase. Les besoins peuvent se traduire sous plusieurs formes:
- des spécifications générales,
- des spécifications fonctionnelles,
- des spécifications d'interface.

Les spécifications générales sont un ensemble d'objectifs, de contraintes (utilisation de matériels et outils existants) et de généralités qu'il faudra respecter au cours du développement.

Les spécifications fonctionnelles sont la description des fonctionnalités du logiciel de manière aussi détaillée que nécessaire.

Les spécifications d'interface sont la description des interfaces du logiciel avec le monde extérieure (hommes, autres logiciels, matériels...) de manière aussi détaillée que nécessaire.

Les spécifications des besoins servent à définir ce que doit faire le logiciel et non comment il est fait. Ceci est décrit dans le document de spécifications des besoins.



Les besoins existants

Spécifications
des besoins du logiciel

Le but de la première phase de développement est de spécifier
les besoins du logiciel à partir des besoins existants

Les moyens et techniques utilisés pour produire le logiciel sont étudiés en parallèle et sont décrits dans le plan de développement du logiciel qui expose l'organisation du projet (décomposition en tâches, structure des équipes, planification, plan de documentation, tests et recette, évaluation de la qualité, répartition des machines de développement et machines cibles).

La spécification des tests de recette sont regroupés dans le cahier de recette du logiciel.

3.1.2. Divers

Cette phase qui constitue environ 15% du temps total du développement se termine par la revue des spécifications fonctionnelles.

Cette phase peut aussi s'appeler "spécifications externes" ou "analyse des besoins" selon le vocabulaire employé.

3.1.3. Documents produits

Les documents produits au cours de cette phase sont:

- Plan de développement du logiciel
- Spécifications des besoins du logiciel
- Cahier de recette

3.1.4. Méthodes de spécification

Il est possible d'utiliser plusieurs outils de spécifications lors de cette phase.

SADT est une méthode de spécification par décompositions successives des activités ou des données du logiciel. (cf les références).

MERISE est une méthode de développement dont les premières phases permettent de spécifier les fonctions et les données du logiciel. (cf les références).

N.B. Il peut être préférable de détailler le système par ses données du système, les actions effectuées sur ces données et leurs relations plutôt que de détailler les fonctions du système, c'est une approche développée dans la méthode MERISE.

3.2. Phase de conception générale

3.2.1. Objectif

Une première étape dans le processus de conception d'un logiciel, à partir des spécifications des besoins, permet de se focaliser sur la définition de l'architecture du logiciel.

La phase de conception générale permet d'envisager plusieurs solutions au problème posé et d'en étudier leur faisabilité. Pour chaque solution, les choix effectués sont notés avec leurs raisons de façon à distinguer les contraintes réelles du projet des contraintes déduites trop hâtivement. La solution répondant le mieux aux besoins exprimés est retenue et figée.

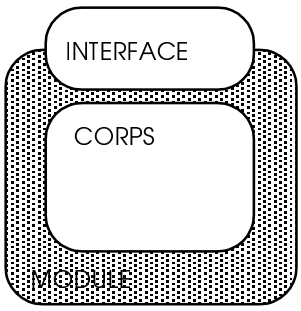
On peut faire des prototypes des différentes approches envisagées dans la conception générale afin de les valider.

Le document de conception générale du logiciel décrit la solution retenue. Une présentation générale de la structure du logiciel est faite.

La structure d'un logiciel peut être vue sous deux points de vue différents:
- le point de vue statique,
- le point de vue dynamique.

3.2.2. Le point de vue statique

Le point de vue statique consiste à découper le logiciel en modules si l'on utilise un langage classique ou de définir les objets du systèmes si l'on utilise un langage orienté objet.

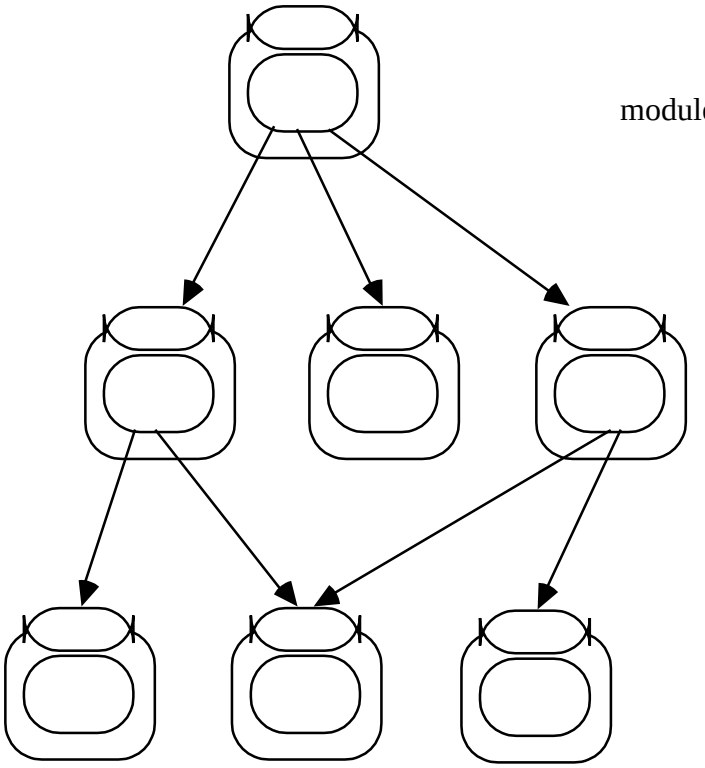


Un module est une unité de compilation regroupant un ensemble de fonctions ou procédures, de définitions de structures de données, de données.

Un module est composé de deux parties:
- une interface visible des autres modules qui peuvent utiliser ce module
- un corps caché et invisible des autres modules.

Un module a la visibilité
- de son corps et de son interface évidemment,
- mais aussi des interfaces des modules qu'il utilise.

exemple de hiérarchie de modules



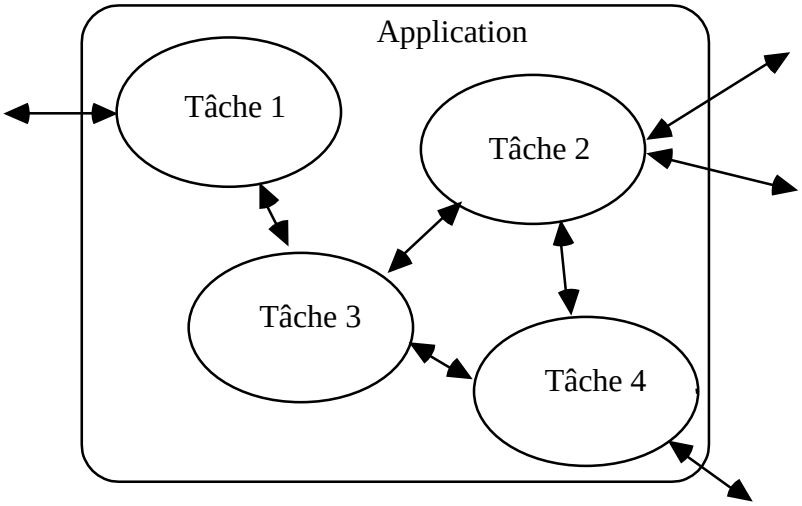
L'interface du module constitue en quelque sorte le contrat que le module s'engage à respecter. Le corps du module sert à remplir ce contrat.

Le rapprochement entre les concepts de module et d'objet peut être le suivant:
Dans un module on essaie de regrouper des fonctions, procédures et données qui ont quelque chose de commun où sont relatives au même concept.
Dans un objet on regroupe des méthodes et des attributs qui ont en commun l'objet lui-même.

3.2.3. Le point de vue dynamique

Le point de vue dynamique consiste à découper le logiciel en tâches pouvant s'exécuter en parallèle (dans le cas où le multitâches est nécessaire). Une tâche est l'exécution d'un code contenu dans un ou plusieurs modules.

Exemple de découpage dynamique d'une application



(Les doubles flèches sont des interfaces de communication intertâches)

La découpe en tâches permet de découpler des traitements indépendants et de les simplifier. La découpe en tâches est donc guidée par l'aspect applicatif du logiciel.

Si des choix d'outil de développement ont été précisés dans les spécifications générales, la découpe en tâches peut dépendre de ces choix (par exemple pour une application multi-langages).

Les deux points de vue (statique et dynamique) permettent de définir la structure du logiciel et sont indispensables à une bonne conception de logiciel.

3.2.4. Divers

La phase de conception générale peut dépendre des outils de développement si ceux-ci ont été spécifiés dans les spécifications générales. Dans le cas où plusieurs outils sont utilisés, il faut trouver comment les utiliser pour réaliser telle partie du système et comment les interfacer. Les découpes statiques et dynamiques dépendent alors aussi de ces outils.

Si aucun outil de développement n'a été spécifié dans la phase précédente, le choix des outils peut être adapté au type de l'application et simplifier les découpes statiques et dynamiques.

Cette phase qui représente environ 10% du temps total consacré au développement se termine par la revue de conception générale.

Cette phase peut aussi s'appeler "conception globale" ou "analyse organique globale" selon le vocabulaire employé.

3.2.5. Document produit

Le document produit au cours de cette phase est:

Document de conception générale

3.2.6. Méthodes de conception

Il est possible d'utiliser une méthode de conception lors de cette phase.

HOOD est une méthode de conception orientée objets (cf les références).

HOOD met l'accent sur les points de vue statique et dynamique. Pour HOOD, l'aspect statique est représenté par la hierarchie senior/junior et l'aspect dynamique est représenté par la hierarchie parent/enfant et dans la définition d'objets actifs.

Certains langages de développement ont été conçus ou inventés non seulement dans l'optique codage mais aussi dans l'optique conception de logiciel. Ada, LTR3 et tous les langages orientés objets dont CRL.

Certains langages ne sont pas modulaire ou orientés objets mais il est fortement conseillé de trouver des palliatifs. Prenons le cas de Unix et C. Un module sera un couple de deux fichiers, le .c sera le corps et le .h l'interface. Le .c pourra inclure le .h de son module et les .h des modules avec lequels il est en relation. Le .h ne comprendra aucun corps de fonctions, seulement des déclarations de fonctions ou de variables.

3.3. Phase de conception détaillée

3.3.1. Objectif

Une deuxième étape dans le processus de conception du logiciel permet, à partir du résultat de la conception générale de pousuivre le découpage du logiciel jusqu'à arriver à une description externe de chacune des procédures et des structures de données.

Dans le cas de l'utilisation d'un langage modulaire, cette phase consiste à définir précisément les interfaces des modules.

Dans le cas de l'utilisation d'un langage orienté objet, cette phase consiste à définir précisément les contenus des objets: attributs et méthodes.

Le document de conception détaillée présente l'architecture détaillée à laquelle on aboutit.

Le manuel d'utilisation du logiciel décrit la mise en oeuvre du logiciel et son environnement.

Les tests et jeux d'essais à mettre en oeuvre durant le phase d'intégration du logiciel sont décrits dans le document de spécifications des tests d'intégration du logiciel.

3.3.2. Divers

On peut aussi décrire le contenu des procédures grâce à un pseudo-langage.

Cette phase qui représente environ 25% du temps total consacré au développement se termine par la revue de conception détaillée.

Cette phase peut aussi s'appeler "Conception détaillée" ou "analyse organique détaillée" selon le vocabulaire employé. Les deux phases de conception peuvent être regroupées sous le vocable de "spécifications internes" selon les cas.

3.3.3. Documents produits

Les documents produits au cours de cette phase sont:

- Document de conception détaillée
- Manuel d'utilisation
- Spécifications des tests d'intégration

3.4. Phase de codage et tests unitaires

3.4.1. Objectif

Les procédures identifiées lors de la phase précédente sont codées et testées individuellement. Le produit de cette phase est le code source et les résultats des tests unitaires.

Dans le cas de l'utilisation d'un langage modulaire, cette phase consiste à coder les corps des modules en respectant leur interface.

3.4.2. Divers

Cette phase qui représente environ 15% du temps total consacré au développement se termine par la revue de codage et tests unitaires.

3.4.3. Documents produits

Les documents produits au cours de cette phase sont:

- Source
- Résultats des tests unitaires

3.5. Phase de test des modules

3.5.1. Objectif

Chaque module est testé individuellement. On vérifie que les services spécifiés par l'interface d'un module sont effectivement rendus par le module.

3.5.2. Divers

Cette phase qui représente environ 5% du temps total consacré au développement se termine par la revue de tests des modules. Cette phase peut éventuellement être fusionnée avec les tests unitaires de la phase précédente.

3.5.3. Documents produits

Les documents produits au cours de cette phase sont:

Résultats des tests de modules

3.6. Phase d'intégration d'ensemble

3.6.1. Objectif

Les différents modules du logiciel sont progressivement intégrés par niveaux successifs en respectant les spécifications des tests d'intégration. La phase d'intégration ressemble à une construction où chaque brique de base du logiciel est associée à sa voisine pour former une entité elle-même associée à sa voisine etc... jusqu'à aboutir à la construction toute entière.

Un premier niveau consiste à tester que des modules liés par une relation d'utilisation fonctionnent correctement ensemble.

Le niveau suivant consiste à tester qu'une tâche fonctionne correctement en utilisant les modules sur lesquels elle s'appuie.

Le niveau suivant consiste à tester que plusieurs tâches liées par des mécanismes multi-tâches (signalisation, exclusion mutuelle, mémoire partagée et plus généralement communication) fonctionnent bien ensemble.

Enfin, le niveau final consiste à mettre en commun toutes les tâches du logiciel pour le tester globalement.

Les jeux d'essais, procédures et résultats des tests sont consignés dans le document de résultat des tests d'intégration.

Un document regroupe les procédures à suivre pour passer du code source au code objet du logiciel.

3.6.2. Divers

Cette phase qui représente environ 20% du temps total consacré au développement se termine par la revue d'intégration d'ensemble du logiciel.

3.6.3. Documents produits

Les documents produits au cours de cette phase sont:

- Présentation du logiciel
- Résultats des tests d'intégration

3.7. Phase de recette

3.7.1. Objectif

Le logiciel lui-même est intégré dans l'environnement extérieur (autres logiciels, utilisateurs). On vérifie que le logiciel développé répond aux besoins exprimés dans la phase de spécifications des besoins.

3.7.2. Divers

Cette phase qui représente environ 10% du temps total consacré au développement se termine par la revue finale.

3.7.3. Documents produits

Les documents produits au cours de cette phase sont:

- Résultats de la recette

4. LA MAINTENANCE

La maintenance est une activité qui comprend la formation de l'utilisateur et l'assistance technique. Elle débute à la livraison du logiciel et s'achève à la fin de l'exploitation du système.

La maintenance peut être:
- corrective: non conformité aux spécifications, d'où détection et correction des erreurs résiduelles.
- adaptative: modification de l'environnement (matériel, fournitures logicielles, outil, ...)
- évolutive: changement des spécifications fonctionnelles du logiciel.

L'activité se prépare pendant le développement et s'applique ensuite sur le logiciel opérationnel recetté.

Lorsque les modifications représentent une partie notable du développement, on les considère comme une refonte sortant du cadre de la maintenance, traitée comme un projet logiciel normal.

Les activités de maintenance couvrent les domaines suivants:
- qualifications des nouvelles versions,
- suivis des modifications,
- archivage,
- mise à jour de la documentation,
- exécution des modifications.